下载APP

答疑课堂｜爬虫反爬虫调试对抗技巧以及虚拟机详解

2022-02-23 DS Hunter

《反爬虫兵法演绎20讲》 课程介绍 >

讲述：DS Hunter
时长 18:02 大小 16.52M

你好，我是 DS Hunter。

因为一些保密需要，我们的课程充满了理论，可能你会认为难以落地。因此啊，我们总要有一讲，擦着泄密的边缘，稍稍讲一点实际的例子，也在课程的基础与理论实践篇快要结束的时候做个总结。

反爬虫领域里的例子就像魔术揭秘一样，一旦了解背后的秘密了，我打赌你会先惊讶一下：哦原来是这样？然后仔细想想，就会再说：哦……不过如此！

不过在后续自己做的时候，还是要咬着牙慢慢自己往下 Coding 才行，你会有一种“感觉看了一堆明明能直接落地的东西，但无能为力”的感觉，在自己落地试试的时候，似乎也摔得不轻。

是的，这就是反爬。相同的招式对圣斗士是无效的，我们必须自己学会创新才行。我们能做的，就是看历史、学思路。

今天，我会为你总结技术对抗与非技术对抗这两大思路，并一一展开说明。最后，也会给你讲一讲在反爬虫中加密强度最高、最复杂的大招——虚拟机。也就是，对各类反爬虫的手段以及实现方法做一个快速的总结。

技术对抗

首先我们要明确，所有技术对抗的本质目标不是加密，而是浏览器 bug 检测。其余的所有手段都是为了隐藏自己的这个目标。

我们提到过，假币唯一的特征就是与真币不一样。那么爬虫唯一的特征就是与浏览器不一样。哪里不一样？浏览器有 bug，爬虫不一定有。你要模拟得一样，需要模拟所有的 bug 才行。我们都知道，抄别人代码很容易，但是抄别人 bug？那可不是一般的困难。

举个例子，我们看这样一个小 bug：

> 在部分版本的 IE8 下面，如果一个 a 标签的 innerText 与 href 相同，那么当你修改 href 的时候，它会自动帮你修改 innerText。
>
> 这个很傻的 bug 意味着什么？
>
> 这意味着你作为一个爬虫，如果对网站声称自己是 IE8，那么，他会让创建一个 a 标签，然后运行相关的 js 操作 DOM。然后，如果你能把这个 bug 模拟对了，他就相信你是浏览器。如果你不知道有这个 bug，你妥妥的按照正常的方式去模拟。那么，你就是爬虫！

但是，请注意，如果你代码赤裸裸写着：if，你是 IE8，那么，你要创建一个 a 标签……等等。你觉得爬虫傻吗，他们不会觉得这个有问题吗？他们看你这么弄肯定觉得奇怪，会真的装个 IE8 试试啊。然后调试完了，他发现，什么，原来是这样，太阴险了。下次他还会上当吗？

想让一个人上两次当，唯一的办法就是：让他不知道自己上当了。这样，你卖完拐，卖完车，改个担架还能卖给他。

那么，强加密必然是基操了。诶，是不是你这时候忽然意识到了 js 加密的重要性？而且是不是这时候也明白了，加密不是目标，加密是手段？浏览器 bug 的检测才是目标。**或者说，各个浏览器的差异，才是反爬虫技术对抗的核心。**

好了，还是要帮你解决一下这个问题，我们回到这个例子本身。你可能会觉得：我知道这个 bug 了，可是要怎么知道哪些版本的浏览器会触发这个 bug 呢？注意，不要一个一个检测。有些人可能会有洁癖，一定要知道 IE 什么版本，UA 里面拆除一堆字段，然后逐个判断。注意啊，不需要！

我们可以直接用浏览器 bug 和 UA 进行 mapping！也就是说，有 bug 的所有 UA 如果我都能存起来，这个问题是不是就解决了？我不需要拆解 UA 啊。那么如何建立这个 mapping 呢？实际上，你只要预先用这个规则跑一段时间，但是不拦截，直接把埋点发送回来就可以了。一般做反爬的公司，都不是什么小厂，有足够的流量来做这个操作。线上跑个一周半周就可以了。

那么，你可能会问了：那如果对方发现我检测这个，不是提前就有准备了吗！

没错！那么你再想想怎么办才好？

哦，是不是给埋点设置一个强加密就行了？这个破解了也不影响拉接口，爬虫才没心思破解这玩意呢。这时候你是不是就意识到了埋点需要强加密的原因了？

到这里，你应该明白了，反爬的战场是瞬息万变的。很多经验，都是由实战训练出来的；很多选择，等你面对的时候才知道要怎么选。如果你觉得前面有些结论似乎没什么用，不妨反过来想想：如果我反向操作会引发什么问题吗？当你想到了问题所在，自然就明白为什么要这么操作了。

下面我们看几个例子。

迭代对抗：hook 与反 hook

hook 这个技术本来是在 native 世界常用的方法。但是浏览器端，因为 js 修改的便利性，hook 的玩法更加丰富多彩。

我们看这样一段对话：

> A：我的接口需要检测 xx 信息，并且经过了超强的加密，你信息不对我就不给你调接口！
>
> B：我根本不用看你的加密逻辑。我用 xx 工具进行操作，hook 掉你的检测接口，如果你检测，我就 mock 一个数据给你，让你以为自己拿到了物理数据，实际是我模拟的！
>
> A：Too young。我用 API 检测你的软件列表，如果你安装了 xx 工具，我就不让你运行！
>
> B：你才 too young，我 hook 你的检测 API，你问我装没装这个工具，我就说我没装！
>
> A：你才 too young，我用 native 代码实现检测，然后用框架调用，让你无法在框架级别进行 hook！
>
> B：你才 too young，我 hook 你调用 native 的方法，然后塞给你一个我写的函数，说这就是你的 native 方法！
>
> ……

其实，这个对话还可以不断的扩展下去，这里先赶紧打住。我们可以看到，欺诈与反欺诈检测是一个无休止的问题，只要一方解决了对方出的难题，就可以立刻进行反制。

而大家经常说的：**反爬虫系统最终总是能破解，只是时间问题。**这句话是什么意思呢？其实这句话你可以这么理解：反爬虫系统，对于爬虫来说，相当于一个单机游戏。单机游戏，总有通关的一天。也许是明天，也许是后天，也许是下个月、半年？但是总是过得去的。甚至可以开外挂来过啊。

那么什么游戏不能通关呢？没错，网络游戏不能通关，它可以无休止地升级下去。你装备刚毕业，他改版了。就这样，一直下去，等于没有破解。就像一些模拟战争游戏抢要塞一样，刚抢完没多久，过期了，又要重抢。这就等于没怎么抢到手。

那么对应 js 里面，这个是什么呢？其实就是对函数的拦截、模拟、切片等等。比如上一课我们提到的 eval 的攻防，就是一个简单的 hook 与反 hook。

此外，其余的函数也可以用这种互相 hook 检测的办法。这个和石头剪子布一样，永远是最后出的赢。我们要做的，就是不断地更新，迭代改版。

Debugger 对抗：无限 Debugger

反爬虫方反爬虫方针对很多函数，都会设置大量的 debugger，甚至在 EventLoop 里不断设置。

事实上，这样的办法是拦截初级调试者的。爬虫方呢，只要在代理上进行一次拦截，然后批量替换掉所有的 debugger 即可。这样，当 js 到达浏览器的时候，debugger 就完全消失了。

但是，反爬虫方也可以针对这种批量替换继续出招。例如，检测函数的 toString 是否包含 debugger。如果不包含，就可以认定函数被篡改。事实上，判定函数的 toString，是反爬虫方一个很常用的技巧。在一定程度上，这甚至可以当函数签名使用。而一旦有了这样的代码，反爬虫方就不敢随意替换了。

注意：debugger 甚至不一定放在代码里，还可以放在注释里。要知道，Function 的 toString，是包含注释在内的。这样，不但可以检测 debugger 在不在，还可以检测 debugger 的位置，并且不影响自己调试。另外有了这个思路之后，注释内可以隐藏大量的签名。

视觉对抗：阿拉伯文与 emoji

很多浏览器在指定区域语言下，多语言支持做得是比较差的。如果线上测试发现指定的浏览器版本支持多语言，并且兼容很好，那么阿拉伯语将成为一个相当恐怖的杀器。

类似这类语言的字符有个很恐怖的地方，就是它们有的时候是从右向左写的。而且对于不懂阿拉伯语的人来说，根本不理解到底为啥代码左右乱跳。此外，一些畸形的字符都可以使用，都可以在视觉上严重影响阅读。例如 QQ 群里常见烟斗字符等等。

此外，emoji 也是一个十分影响观感的一个选项。举个例子，10 个狗构成一个变量名，11 个狗构成另一个变量名，我相信正常人类是很难在一群狗里面调试下去代码逻辑的。当然，这里同样只是建议轻微使用，不建议大规模铺开。

理由有两个。一方面是兼容性问题。一旦出现，非常难调试。理论上它们可以做变量名，但是实际上哪个浏览器兼容，要以线上测试为准。另一方面，对方只要再走一次 minify 即可，这个我们前面提到过，破解成本比较低，所以我们大规模铺开的意义不大。因此，emoji 这种方法不宜做大招使用。不过，小范围恶心一下对手，也很可能成为压死骆驼的最后一根稻草。

JSFuck 对抗

大名鼎鼎的JSFuck，这个发明一度震惊了全世界。js 的混乱程度被用到了极致。但是要注意，这种加密方式已经有比较成熟的工具来逆向解析了。此外，代码长度急剧膨胀，简直不能忍受。因此，如果使用，只能放在核心代码上绕一下对手。而且，建议进行自定义，而不是使用标准做法，避免被工具逆向。

工具对抗：调试状态检测

判定浏览器调试状态也是一个很关键的做法。在一些浏览器里，你可以有意引发一个异常然后 catch 住，判定 error 的信息。你会发现，在 Console 里运行的函数与直接运行的函数，引发的 error 有很大差别，这个差别很明显，就不详细展开了。此外，启动 Devtools 的时候会引发一次 size 的 change，这个也可以持续监听。最后，Console 里运行的时候，会有一些 Console 特有的函数。

这些都可以做判定。具体可以查阅 Chrome 的帮助文档。

非技术对抗

除了正面硬刚的技术对抗，还会有一些角落中的非技术对抗。非技术对抗大部分是心理对抗。当然，在很大程度上，心理对抗都比技术对抗效果好。毕竟，反爬打的不是算法，是人。

薪资对抗

薪资对抗，或者福利对抗，都是比较常用的心理战术。例如，随着调试的不断深入，不断给对方报价。

举个例子，爬虫方破解你第一层加密的时候，提示：唉，这算啥啊，月薪三位数都能走到这。再破解一层，就提示：这也就拿个月薪四位数。甚至可以直接报价，乃至于直接留 HR

邮箱。这并非天方夜谭，很多爬虫工程师技术非常优秀，并且非常不想继续做爬虫了，能转前端多开心啊！专业也对口，HR 业绩也完成了。

但是注意：不要在这里鄙视竞对，否则容易惹上法务风险。

关键字的作用

很多人都知道，某知名网站在反爬里经常会写一些稀奇古怪的话。这些话非常猖狂，让人破解完忍不住要写一篇博客吐槽他们。注意，你写的博客很快会被搜索引擎收录。而他们有个爬虫帮自己不断检测关键字是否被收录。嗯，没错，你被发现了，爬取时自以为的低调真的只是“自以为”了！他们反爬组当然会写爬虫啊！

因此后续记住，看到反爬代码里的冷嘲热讽，或者一些稀奇古怪的话，无视即可，千万不要当回事。只要多看一眼，你可能就输了。

重头：虚拟机

很多大型反爬一定会有一个虚拟机的。但是回到我们刚刚说到的反爬虫思路来看，虚拟机不是目的，只是手段。webassembly 也是虚拟机。但是我们在第 10 讲提到过，webassembly 不好用，大家还是会自己实现虚拟机的。而**虚拟机的核心作用，就是对代码进行加密，让破解变得更难。**因为只有难以破解的代码，才能更好地隐藏你的小心机。

虚拟机的实现

虚拟机的实现方式，基本上就是看你熟悉哪门语言，就实现对应的解释器就行了。一般不建议使用主流语言，而是尽可能用冷门语言。Lisp 或者汇编都是不错的选择。甚至，你自创一门语言，都是可以的。只要你有办法写完解释器。

解释器本身还需要再进行加密。例如，eval，变量名混淆，都可以。甚至更疯狂地说，我们都知道大部分语言都是自举的。也就是说，你的解释器可以用另一个解释器来解释执行，套娃解释。最终，你的竞对一定会被绕晕，根本看不出这是两层解释器。

至于再增加层数，就没必要了。理由是，两层解释器的代码已经大大超出预期。再高可能会因为 js 过大，影响用户体验了。

这里强烈安利 Lisp 解释器。我不方便透露太多细节，我只能说，某站点的反爬是用 Lisp 解释器做的，已经跑了六七年了，直到现在，我搜各种破解博客，大家还都在纠结如何破解那个“汇编解释器”的问题。事实上，那是 Lisp……他只是简单解释了一颗语法树……

虚拟机的部署

实现了虚拟机的功能，我们就要开始部署了。虚拟机的部署有站点直接部署与 CDN 部署两种。

首先看站点部署。最原始的部署方式一定是由站点直接生成 js 文件，那么文件头就会有一个虚拟机。因为虚拟机是共用的，所以有些大聪明可能会考虑放到 CDN。也不是不能放 CDN，但是放了 CDN，你的虚拟机就定死了，不能变更了。实际上，这样反而降低了别人的破解难度。

但是不放 CDN，又太慢了，因为很难使用缓存机制。怎么办呢？

实际上，如果你是两层虚拟机，那么可以把一层放在 CDN，利用缓存。然后再套娃解释一个简单虚拟机。例如，CDN 放汇编解释器，然后用汇编解释器解释一个 Lisp 解释器，Lisp 解释器用站点直接下发。这样，就可以实现 CDN 大量加速，同时代码破解难度也不低。

在这种情况下，我们的代码复杂度会逐渐提升。那么，如果爬虫方不读代码，直接用浏览器运行呢？

谢天谢地那太好了，我们等得就是这一天。一旦爬虫不读代码了，那就意味着，无论发生什么，都是黑盒了，他失去了调试的能力。你可以随机封他 IP，封他指纹，增加浏览器 bug 检测。放心检测就好，虽然他使用了浏览器，但他还是会不断换 UA 的，假装自己是很多种浏览器的，所以你可以用各种办法随机坑他。而他只会认为自己的浏览器模拟有问题，而不是去调试哪行代码有问题——毕竟他是自己放弃这个能力的。

这能怪谁呢……

好了，今天的答疑课堂就到这里了。希望我的分享可以帮助你了解反爬虫这件事上思路的重要性。而当你没有思路的时候，一定要回到最初的想法：我完成反爬虫这个动作的最终

目的是什么呢？如果我反向操作会引发什么问题吗？**回归初心，往往能够帮助你直达目标。**

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

生成海报并分享

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 11 | 反爬虫概述（三）：前端反爬虫是怎么做信息收集的？

精选留言 (3)

写留言

leslie
2022-02-23

"回归初心，往往能够帮助你直达目标。"做与不做，皆要问此。

1

一步 新

2022-02-24

利用浏览器的bug来进行反爬，这就需要对各种浏览器的特征非常熟悉，而且还是不常用的特征

作者回复: 线上bug库都用一遍就行，出题者占优势，答题者劣势。

peter
2022-02-24

请教老师几个问题：
Q1：虚拟机是类似于JVM，用来运行代码的吗？还是用来混淆 js 代码的?

Q2：如果虚拟机是用来运行代码的，那一般是运行在服务端，爬虫怎么能访问虚拟机呢？

作者回复: 不是服务端那种虚拟机，是用js解释的，执行在前端。好多家都有。类似jvm，不是vmware那种虚拟机。